



8-Puzzle Solver Game

Trò chơi giải đố 8-puzzle với nhiều nhóm thuật toán tìm kiếm AI

python 3.x pygame 2.x License MIT

BÁO CÁO TỔNG KẾT DỰ ÁN

1. Mục Tiêu

Mục tiêu của dự án 8-Puzzle Solver Game là phát triển một trò chơi giải đố 8-puzzle tích hợp nhiều thuật toán tìm kiếm AI, với giao diện trực quan và khả năng trực quan hóa từng bước giải. Cụ thể, dự án hướng đến các mục tiêu sau:

- Xây dựng trò chơi giải đố 8-puzzle với giao diện đồ họa trực quan, cho phép người dùng dễ dàng nhập trạng thái ban đầu, theo dõi trạng thái bàn cờ và tương tác với các thao tác giải thông qua các nút chọn thuật toán và điều khiển trên màn hình.
- Triển khai nhiều thuật toán tìm kiếm thuộc 6 nhóm lớn gồm: tìm kiếm không có thông tin, tìm kiếm có thông tin, tìm kiếm cục bộ, tìm kiếm trong môi trường phức tạp, bài toán ràng buộc (CSP), và học tăng cường. Các thuật toán này được áp dụng trên cùng một bài toán 8-puzzle để so sánh khả năng tìm lời giải, từ đó làm rõ sự khác biệt về logic và hiệu quả giữa các chiến lược AI.
- Phân tích ưu nhược điểm của từng thuật toán khi áp dụng vào bài toán cụ thể, giúp làm nổi bật điểm mạnh, điểm yếu và phạm vi ứng dụng của mỗi thuật toán.
- Trực quan hóa toàn bộ quá trình giải đố thông qua giao diện đồ họa: người dùng có thể quan sát từng bước chuyển đổi trạng thái, số bước thực hiện, các trạng thái trung gian và kế hoạch hành động của mỗi thuật toán, giúp việc học và trình bày trở nên dễ hiểu và sinh động hơn.

2. Nội Dung

2.1. Nhóm 1: Tìm Kiếm Không Có Thông Tin (Uninformed Search)

Thành phần chính của bài toán tìm kiếm:

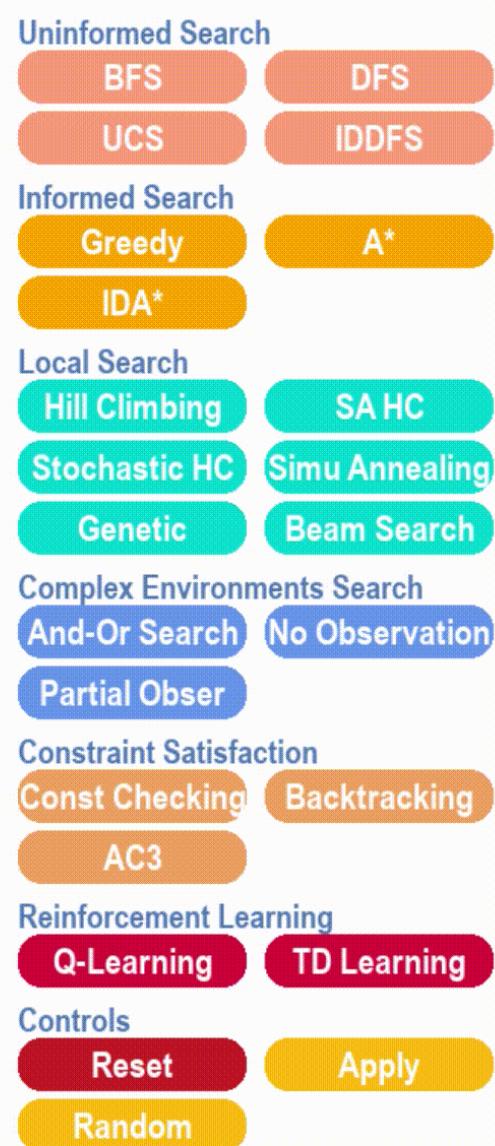
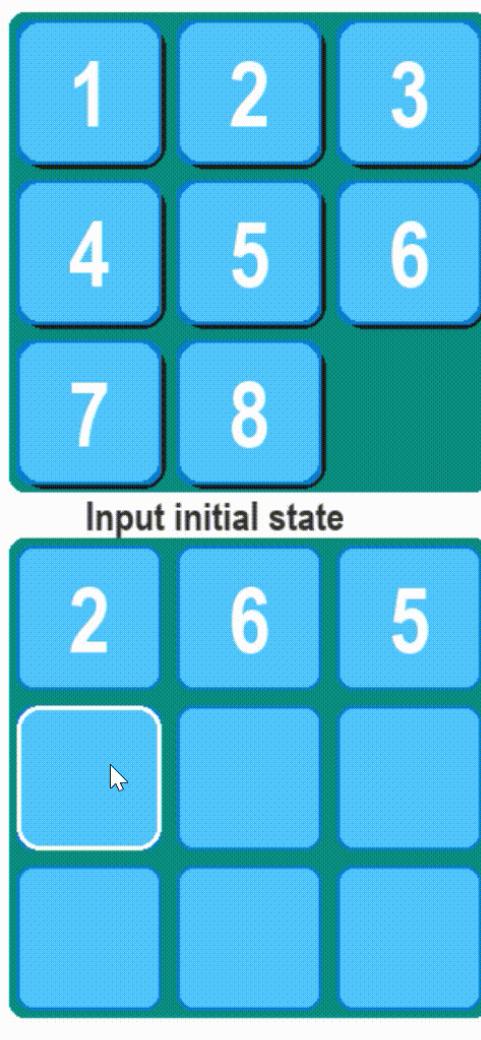
- Trạng thái ban đầu:** Cấu hình khởi điểm của bảng 8-puzzle, do người dùng tùy ý nhập thông qua giao diện.

- **Trạng thái đích:** Cấu hình mục tiêu cần đạt tới, thường được chuẩn hóa là (1, 2, 3, 4, 5, 6, 7, 8, 0), trong đó 0 đại diện cho ô trống.
- **Hàm chi phí:** Mỗi hành động di chuyển giữa hai trạng thái có chi phí bằng 1 đơn vị.
- **Solution:** Chuỗi các bước di chuyển từ trạng thái ban đầu đến trạng thái đích

Các thuật toán tìm kiếm không có thông tin:

BFS (Breadth-First Search) 🔎

- **Mô tả:** Duyệt theo từng lớp, mở rộng tất cả các đỉnh ở cùng độ sâu trước khi đi sâu hơn
- **Minh họa:**

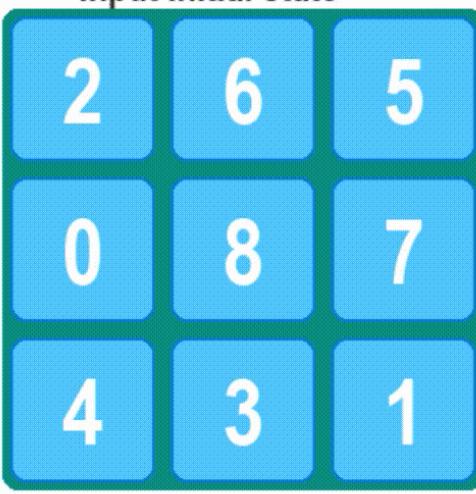


DFS (Depth-First Search) 🕵️

- **Mô tả:** Đi sâu nhất có thể theo từng nhánh trước khi quay lại
- **Minh họa:**



Input initial state



Ready

Uninformed Search
DFS

BFS
UCS
IDDFS

Informed Search
A*

Greedy
IDA*

Local Search

Hill Climbing
SA HC

Stochastic HC
Simu Annealing

Genetic
Beam Search

Complex Environments Search

And-Or Search
No Observation

Partial Obser

Constraint Satisfaction

Const Checking
Backtracking

AC3

Reinforcement Learning

Q-Learning
TD Learning

Controls

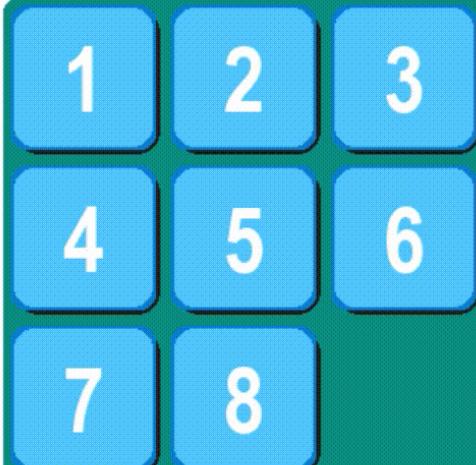
Reset
Apply

Random

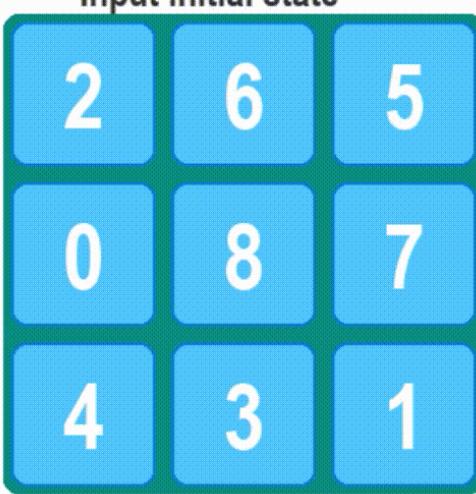
UCS (Uniform Cost Search) ⓘ

- **Mô tả:** Luôn chọn mở rộng nút có chi phí thấp nhất
- **Minh họa:**

3 / 30



Input initial state



Ready

Uninformed Search

BFS	DFS
UCS	IDDFS

Informed Search

Greedy	A*
IDA*	

Local Search

Hill Climbing	SA HC
Stochastic HC	Simu Annealing
Genetic	Beam Search

Complex Environments Search

And-Or Search	No Observation
Partial Obser	

Constraint Satisfaction

Const Checking	Backtracking
AC3	

Reinforcement Learning

Q-Learning	TD Learning
------------	-------------

Controls

Reset	Apply
Random	

IDDFS (Iterative Deepening Depth-First Search) [🔗](#)

- **Mô tả:** Kết hợp DFS và BFS bằng cách lặp DFS theo từng mức độ sâu tăng dần
- **Minh họa:**

Input initial state

1	2	3
4	5	6
7	8	

Ready

Uninformed Search

- BFS
- DFS
- UCS
- IDDFS

Informed Search

- Greedy
- A*
- IDA*

Local Search

- Hill Climbing
- SA HC
- Stochastic HC
- Simu Annealing
- Genetic
- Beam Search

Complex Environments Search

- And-Or Search
- No Observation
- Partial Obser

Constraint Satisfaction

- Const Checking
- Backtracking
- AC3

Reinforcement Learning

- Q-Learning
- TD Learning

Controls

- Reset
- Apply
- Random

So sánh hiệu suất:

Thuật toán	Thời gian giải (ms)	Bộ nhớ sử dụng	Số bước tối ưu
BFS	250-500	Cao	Luôn tối ưu
DFS	50-100	Thấp	Thường không tối ưu
UCS	150-250	Trung bình	Luôn tối ưu
IDDFS	100-200	Trung bình	Luôn tối ưu

Nhận xét:

- BFS** đảm bảo tìm được đường đi ngắn nhất nhưng tiêu tốn nhiều bộ nhớ khi độ sâu của giải pháp tăng
- DFS** tiết kiệm bộ nhớ nhưng không đảm bảo tìm được đường đi ngắn nhất và có thể rơi vào vòng lặp vô hạn
- UCS** tương tự BFS trong bài toán 8-puzzle (vì mỗi bước di chuyển có chi phí bằng nhau), nhưng hiệu quả hơn trong các bài toán có chi phí khác nhau

- **IDDFS** kết hợp ưu điểm của BFS (đảm bảo tìm được đường đi ngắn nhất) và DFS (tiết kiệm bộ nhớ), nhưng có thể tốn thời gian do phải duyệt lại các nút nhiều lần

2.2. Nhóm 2: Thuật Toán Tìm Kiếm Có Thông Tin (Informed Search)

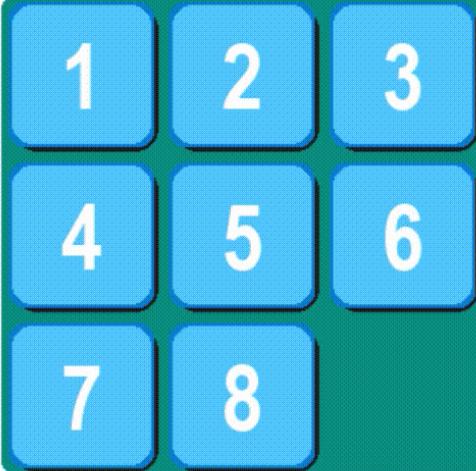
Thành phần chính của bài toán tìm kiếm:

- **Trạng thái ban đầu:** Cấu hình khởi điểm của bảng 8-puzzle, do người dùng tùy ý nhập thông qua giao diện.
- **Trạng thái đích:** Cấu hình mục tiêu cần đạt tới, thường được chuẩn hóa là (1, 2, 3, 4, 5, 6, 7, 8, 0), trong đó 0 đại diện cho ô trống.
- **Hàm chi phí:** Mỗi hành động di chuyển giữa hai trạng thái có chi phí bằng 1 đơn vị.
- **Hàm heuristic ($h(n)$):** Ước lượng chi phí còn lại từ trạng thái hiện tại (x_1, y_1) đến trạng thái đích, đóng vai trò dẫn đường cho quá trình tìm kiếm.
 - **Ví dụ heuristic phổ biến:**
 - **Manhattan distance:** Tổng khoảng cách Manhattan của mỗi ô từ vị trí hiện tại (x_1, y_1) đến vị trí đích (x_2, y_2), với công thức là $|x_1 - x_2| + |y_1 - y_2|$.
- **Solution (Lời giải):** Là chuỗi các hành động hợp lệ dẫn từ trạng thái ban đầu đến trạng thái đích với tổng chi phí thấp nhất theo đánh giá của thuật toán.

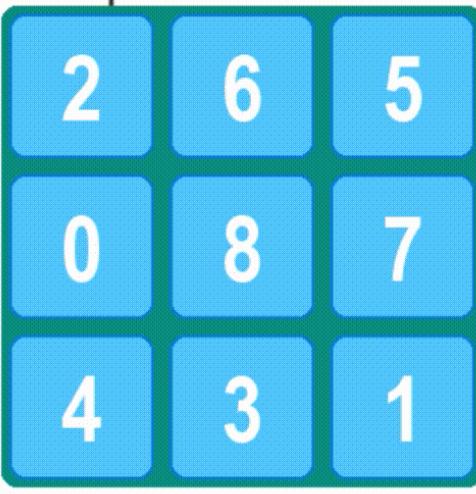
Các thuật toán tìm kiếm có thông tin:

Greedy Search 🌐

- **Mô tả:** Ưu tiên chọn nút gần đích nhất theo hàm heuristic
- **Minh họa:**



Input initial state



Ready

Uninformed Search
BFS
DFS

UCS
IDDFS

Informed Search
Greedy
A*

IDA*

Local Search
Hill Climbing
SA HC

Stochastic HC
Simu Annealing

Genetic
Beam Search

Complex Environments Search
And-Or Search
No Observation

Partial Obser

Constraint Satisfaction
Const Checking
Backtracking

AC3

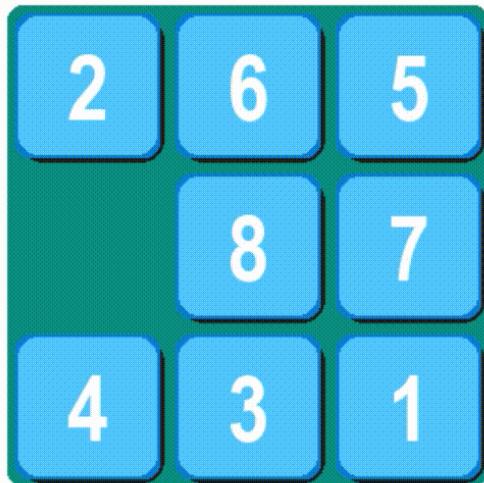
Reinforcement Learning
Q-Learning
TD Learning

Controls
Reset
Apply

Random

A* (A Star) ★

- **Mô tả:** Kết hợp giữa chi phí thực tế ($g(n)$) và chi phí ước lượng đến đích ($h(n)$) để tìm đường tốt nhất
- **Minh họa:**



Ready
Algorithm: Apply



IDA* (Iterative Deepening A Star) 🔑

- **Mô tả:** Phiên bản tiết kiệm bộ nhớ của A*, thực hiện theo tầng
- **Minh họa:**

Input initial state

1	2	3
4	5	6
7	8	

Ready

2	6	5
0	8	7
4	3	1

Uninformed Search

- BFS
- DFS
- UCS
- IDDFS

Informed Search

- Greedy
- A*
- IDA*

Local Search

- Hill Climbing
- SA HC
- Stochastic HC
- Simu Annealing
- Genetic
- Beam Search

Complex Environments Search

- And-Or Search
- No Observation
- Partial Obser

Constraint Satisfaction

- Const Checking
- Backtracking
- AC3

Reinforcement Learning

- Q-Learning
- TD Learning

Controls

- Reset
- Apply
- Random

So sánh hiệu suất:

Thuật toán	Thời gian giải (ms)	Bộ nhớ sử dụng	Số bước tối ưu
Greedy	50-150	Thấp	Thường không tối ưu
A*	100-200	Trung bình	Luôn tối ưu
IDA*	150-300	Thấp	Luôn tối ưu

Nhận xét:

- **Greedy Search** rất nhanh nhưng không đảm bảo tìm được đường đi ngắn nhất
- A* kết hợp hiệu quả giữa UCS và Greedy, đảm bảo tìm được đường đi ngắn nhất nếu heuristic admissible
- IDA* có hiệu suất bộ nhớ tốt hơn A* nhưng có thể chậm hơn do phải duyệt lại các nút

2.3. Nhóm 3: Thuật Toán Tìm Kiếm Cục Bộ (Local Search)

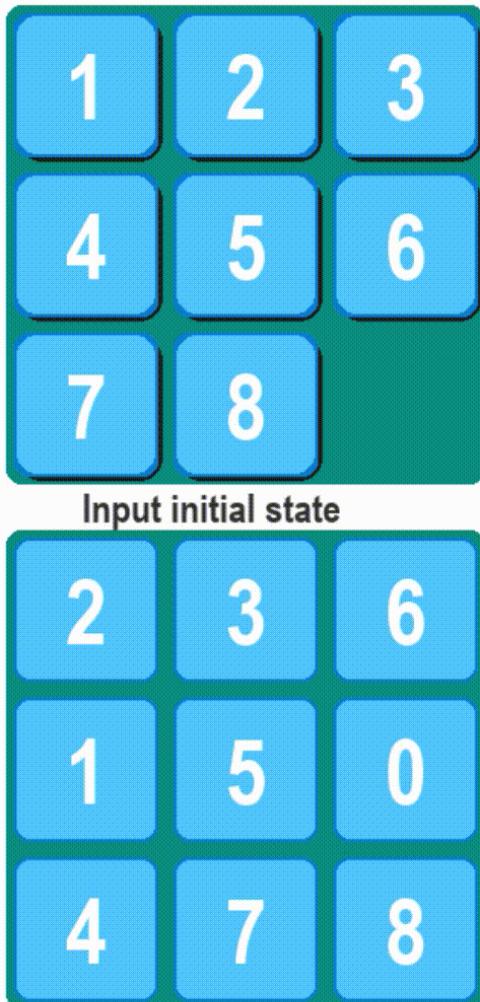
Thành phần chính của bài toán tìm kiếm:

- **Trạng thái ban đầu:** Cấu hình khởi điểm của bảng 8-puzzle, được người dùng nhập tùy ý thông qua giao diện.
- **Trạng thái đích:** Cấu hình mục tiêu cần đạt tới, thường là (1, 2, 3, 4, 5, 6, 7, 8, 0), trong đó 0 đại diện cho ô trống.
- **Hàng xóm (Neighbors):** Tập các trạng thái có thể sinh ra từ trạng thái hiện tại bằng một bước di chuyển hợp lệ. Local Search chỉ xét trạng thái hàng xóm trực tiếp thay vì xây dựng toàn bộ cây tìm kiếm.
- **Hàm đánh giá (Evaluation function):** Hàm dùng để đánh giá "độ tốt" của một trạng thái hiện tại, thường dựa trên khoảng cách đến trạng thái đích.
 - **Ví dụ phổ biến:**
 - **Manhattan distance:** Tổng khoảng cách Manhattan của tất cả các ô (trừ ô trống) từ vị trí hiện tại đến vị trí đúng trong trạng thái đích, tính theo công thức: $|x_1 - x_2| + |y_1 - y_2|$.
- **Solution (Lời giải):** Là một trạng thái gần với mục tiêu hoặc đạt được mục tiêu, được tìm thông qua quá trình cải thiện dần từ trạng thái ban đầu sang trạng thái tốt hơn trong không gian hàng xóm.

Các thuật toán tìm kiếm cục bộ:

Hill Climbing

- **Mô tả:** Luôn di chuyển đến trạng thái tốt hơn nếu có
- **Minh họa:**



Ready



Steepest-Ascent Hill Climbing 🔗

- **Mô tả:** Chọn trạng thái tốt nhất trong tất cả hàng xóm
- **Minh họa:**

Input initial state

1	2	3
4	5	6
7	8	

Ready

Uninformed Search

BFS	DFS
UCS	IDDFS

Informed Search

Greedy	A*
IDA*	

Local Search

Hill Climbing	SA HC
Stochastic HC	Simu Annealing
Genetic	Beam Search

Complex Environments Search

And-Or Search	No Observation
Partial Obser	

Constraint Satisfaction

Const Checking	Backtracking
AC3	

Reinforcement Learning

Q-Learning	TD Learning

Controls

Reset	Apply
Random	

Stochastic Hill Climbing 🎲

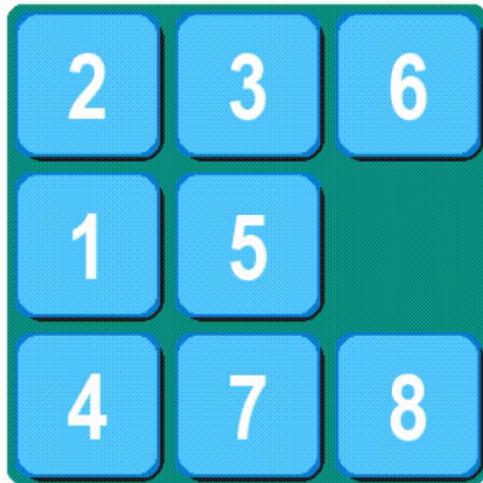
- **Mô tả:** Chọn ngẫu nhiên trong các trạng thái tốt hơn

- **Minh họa:**

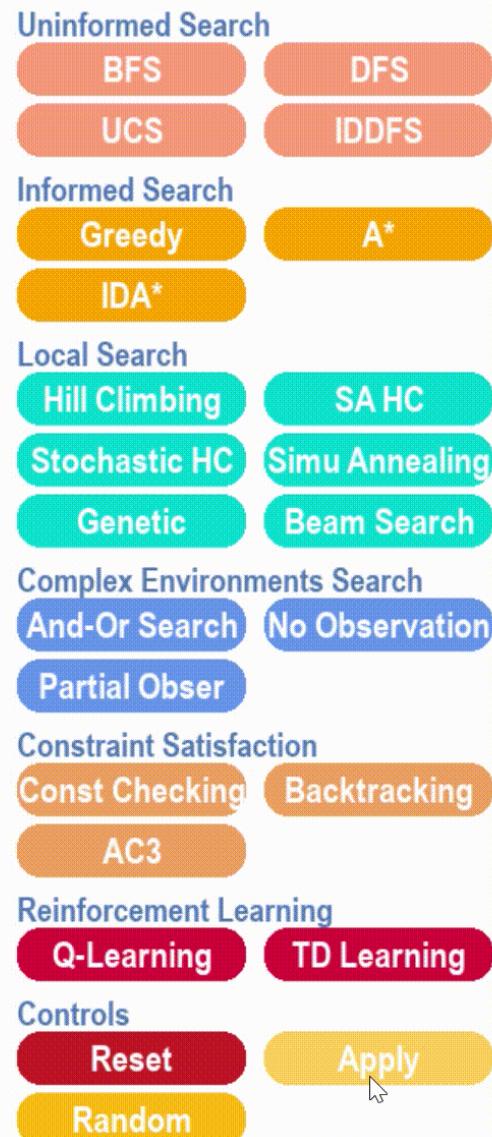


Simulated Annealing ❄️

- **Mô tả:** Chấp nhận trạng thái tệ hơn để thoát khỏi cực trị cục bộ
- **Minh họa:**



Ready
Algorithm: Apply



Genetic Algorithm ↻

- **Mô tả:** Sử dụng các nguyên tắc di truyền để tiến hóa dần đến giải pháp tốt

- **Minh họa:**

Input initial state

1	2	3
4	5	6
7	8	

Ready

2	3	6
1	5	0
4	7	8

Uninformed Search

BFS	DFS
UCS	IDDFS

Informed Search

Greedy	A*
IDA*	

Local Search

Hill Climbing	SA HC
Stochastic HC	Simu Annealing
Genetic	Beam Search

Complex Environments Search

And-Or Search	No Observation
Partial Obser	

Constraint Satisfaction

Const Checking	Backtracking
AC3	

Reinforcement Learning

Q-Learning	TD Learning
------------	-------------

Controls

Reset	Apply
Random	

Beam Search *

- **Mô tả:** Duy trì beam_width trạng thái tốt nhất tại mỗi cấp độ
- **Minh họa:**

Input initial state

1	2	3
4	5	6
7	8	

Ready

Uninformed Search

- BFS
- DFS
- UCS
- IDDFS

Informed Search

- Greedy
- A*
- IDA*

Local Search

- Hill Climbing
- SA HC
- Stochastic HC
- Simu Annealing
- Genetic
- Beam Search

Complex Environments Search

- And-Or Search
- No Observation
- Partial Obser

Constraint Satisfaction

- Const Checking
- Backtracking
- AC3

Reinforcement Learning

- Q-Learning
- TD Learning

Controls

- Reset
- Apply
- Random

So sánh hiệu suất:

Thuật toán	Thời gian giải (ms)	Bộ nhớ sử dụng	Khả năng tìm lời giải
Hill Climbing	30-50	Rất thấp	Có thể bị kẹt
Steepest-Ascent HC	30-50	Rất thấp	Có thể bị kẹt
Stochastic HC	40-60	Rất thấp	Có thể bị kẹt
Simulated Annealing	100-150	Rất thấp	Thường gần tối ưu
Genetic Algorithm	200-350	Trung bình	Có thể tìm ra giải pháp tốt
Beam Search	100-250	Trung bình	Tìm được giải pháp tốt nhất

Nhận xét:

- Các thuật toán **Hill Climbing** rất nhanh và ít tốn bộ nhớ, nhưng dễ bị kẹt ở cực trị cục bộ
- Simulated Annealing** giải quyết được vấn đề kẹt ở cực trị cục bộ nhưng có thể mất nhiều thời gian hơn

- **Genetic Algorithm** đa dạng trong việc tìm kiếm không gian trạng thái nhưng phức tạp hơn và tốn thời gian
- **Beam Search** cho tốc độ tốt nhưng không đảm bảo tìm được đường đi tối ưu nếu beam_width quá nhỏ

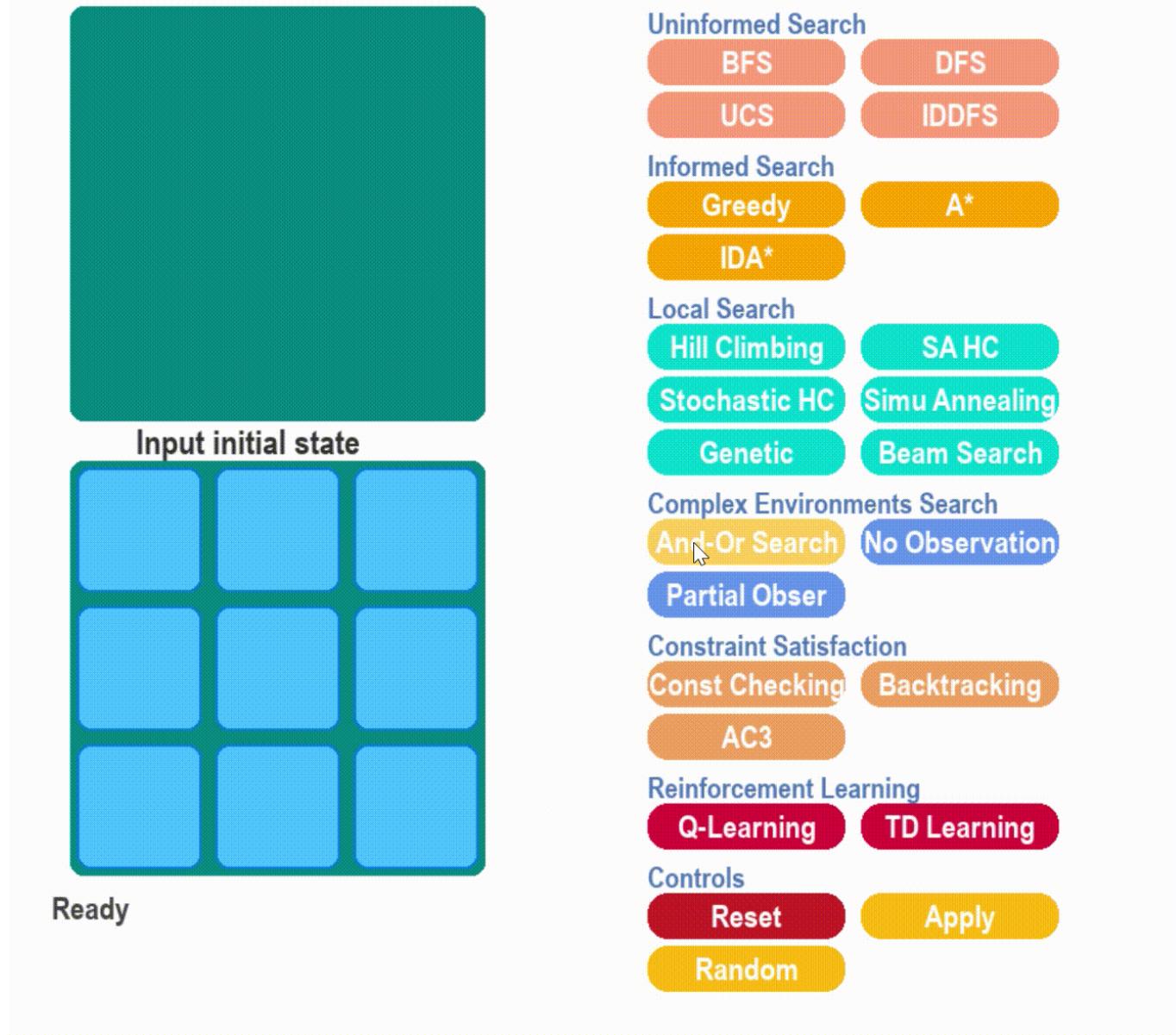
2.4. Nhóm 4: Thuật Toán Tìm Kiếm Trong Môi Trường Phức Tạp

Thành phần chính của bài toán tìm kiếm:

- **Trạng thái ban đầu:** Không còn là một trạng thái xác định duy nhất, mà là một **tập hợp các trạng thái niềm tin (belief state)** do không có đủ thông tin ban đầu.
- **Hành động (Action):** Có thể mang tính **bất định**, nghĩa là một hành động thực hiện từ một trạng thái có thể dẫn đến nhiều kết quả khác nhau, tùy vào điều kiện môi trường.
- **Quan sát (Observation):** Là thông tin gián tiếp thu được sau khi thực hiện hành động, dùng để **cập nhật lại tập hợp belief state** và thu hẹp khả năng nhận diện trạng thái hiện tại.
- **Solution (Lời giải):** Không đơn thuần là một chuỗi hành động tuyến tính, mà là một **kế hoạch có cấu trúc cây (AND-OR plan)** hoặc một chiến lược hành động phù hợp cho mọi khả năng xảy ra, bất chấp việc thiếu thông tin quan sát đầy đủ hoặc môi trường thay đổi không đoán trước.

And-Or Search

- **Mô tả:** Phù hợp cho bài toán có nhiều khả năng lựa chọn và rẽ nhánh
- **Minh họa:**



No Observation Search ⓘ

- **Mô tả:** Giải trong điều kiện không biết rõ trạng thái ban đầu
- **Minh họa:**

Input initial state

2	6	5
0	8	7
4	3	1

Ready

Uninformed Search

- BFS
- DFS
- UCS
- IDDFS

Informed Search

- Greedy
- A*
- IDA*

Local Search

- Hill Climbing
- SA HC
- Stochastic HC
- Simu Annealing
- Genetic
- Beam Search

Complex Environments Search

- And-Or Search
- No Observation
- Partial Obser

Constraint Satisfaction

- Const Checking
- Backtracking
- AC3

Reinforcement Learning

- Q-Learning
- TD Learning

Controls

- Reset
- Apply
- Random

Partial Observable Search

- **Mô tả:** Xử lý bài toán khi chỉ biết một phần trạng thái môi trường
- **Minh họa:**

The screenshot shows a user interface for a search algorithm demonstration. On the left, there's a large green area labeled "Input initial state" containing a 3x3 grid of blue buttons. The grid contains the following values:

1	2	3
6	4	5
0	7	8

Below the grid, the word "Ready" is displayed. To the right of the grid is a vertical column of search algorithm buttons, each grouped under a category:

- Uninformed Search:** BFS, DFS, UCS, IDDFS
- Informed Search:** Greedy, A*
- Local Search:** Hill Climbing, SA HC, Stochastic HC, Simu Annealing, Genetic, Beam Search
- Complex Environments Search:** And-Or Search, No Observation, Partial Obser
- Constraint Satisfaction:** Const Checking, Backtracking, AC3
- Reinforcement Learning:** Q-Learning, TD Learning
- Controls:** Reset, Apply (highlighted with a cursor), Random

So sánh hiệu suất:

Thuật toán	Thời gian giải (ms)	Bộ nhớ sử dụng	Đặc điểm
And-Or Search	300-500	Cao	Tìm giải pháp tối ưu trong cây AND-OR
No Observation	200-400	Thấp	Đưa ra kết quả chính xác trong điều kiện không quan sát
Partial Observable	150-300	Trung bình	Đưa ra kết quả với thông tin quan sát một phần

Nhận xét:

- **And-Or Search** hiệu quả cho các bài toán có nhiều khả năng lựa chọn nhưng tốn nhiều bộ nhớ
- **No Observation** và **Partial Observable** giải quyết được các bài toán với thông tin không đầy đủ
- **Partial Observable** đặc biệt hữu ích trong môi trường mà người giải không thể biết chính xác trạng thái hiện tại, buộc phải dựa vào tập hợp các trạng thái khả thi (belief state) để ra quyết định

2.5. Nhóm 5: Thuật Toán Tìm Kiếm Trong Môi Trường Có Ràng Buộc (CSP)

Thành phần chính của bài toán tìm kiếm:

- **Biến (Variables):** Có tổng cộng **9 biến**, ký hiệu từ **X1** đến **X9**, tương ứng với 9 vị trí trên bảng 3x3 (từ trái qua phải, từ trên xuống dưới).
- **Miền giá trị (Domain):** Mỗi biến nhận một giá trị duy nhất trong tập **{0, 1, 2, ..., 8}**, trong đó 0 biểu thị cho ô trống. Tập giá trị được **xáo trộn ngẫu nhiên** nhằm tăng tính đa dạng khi sinh trạng thái ban đầu.
- **Ràng buộc (Constraints):**
 - **Ràng buộc ngang:** Các cặp ô nằm liền kề theo hàng ngang (ví dụ: **X1-X2, X2-X3, ...**) phải thỏa mãn điều kiện:

Giá trị bên phải = giá trị bên trái + 1, và giá trị bên trái ≠ 0.
 - **Ràng buộc dọc:** Các cặp ô liền kề theo cột dọc (ví dụ: **X1-X4, X2-X5, ...**) phải thỏa mãn điều kiện:

Giá trị phía dưới = giá trị phía trên + 3, và giá trị phía trên ≠ 0.
 - **Ràng buộc toàn cục không trùng lặp:** Mỗi giá trị từ 0 đến 8 chỉ xuất hiện một lần trên toàn bộ bảng (AllDifferent Constraint).
- **Kiểm tra khả năng giải (solvability check):** Sau khi hoàn tất việc gán giá trị cho 9 biến, trạng thái cuối cùng sẽ được kiểm tra tính khả thi bằng hàm **is_solvable()**. Trạng thái chỉ hợp lệ nếu có thể giải được theo luật 8-puzzle.
- **Solution:** Gán giá trị cho 9 biến X1 đến X9, thỏa mãn các ràng buộc (ngang, dọc, không giá trị) và tạo thành một trạng thái có khả năng thực hiện đến trạng thái mục tiêu.

Backtracking ↪

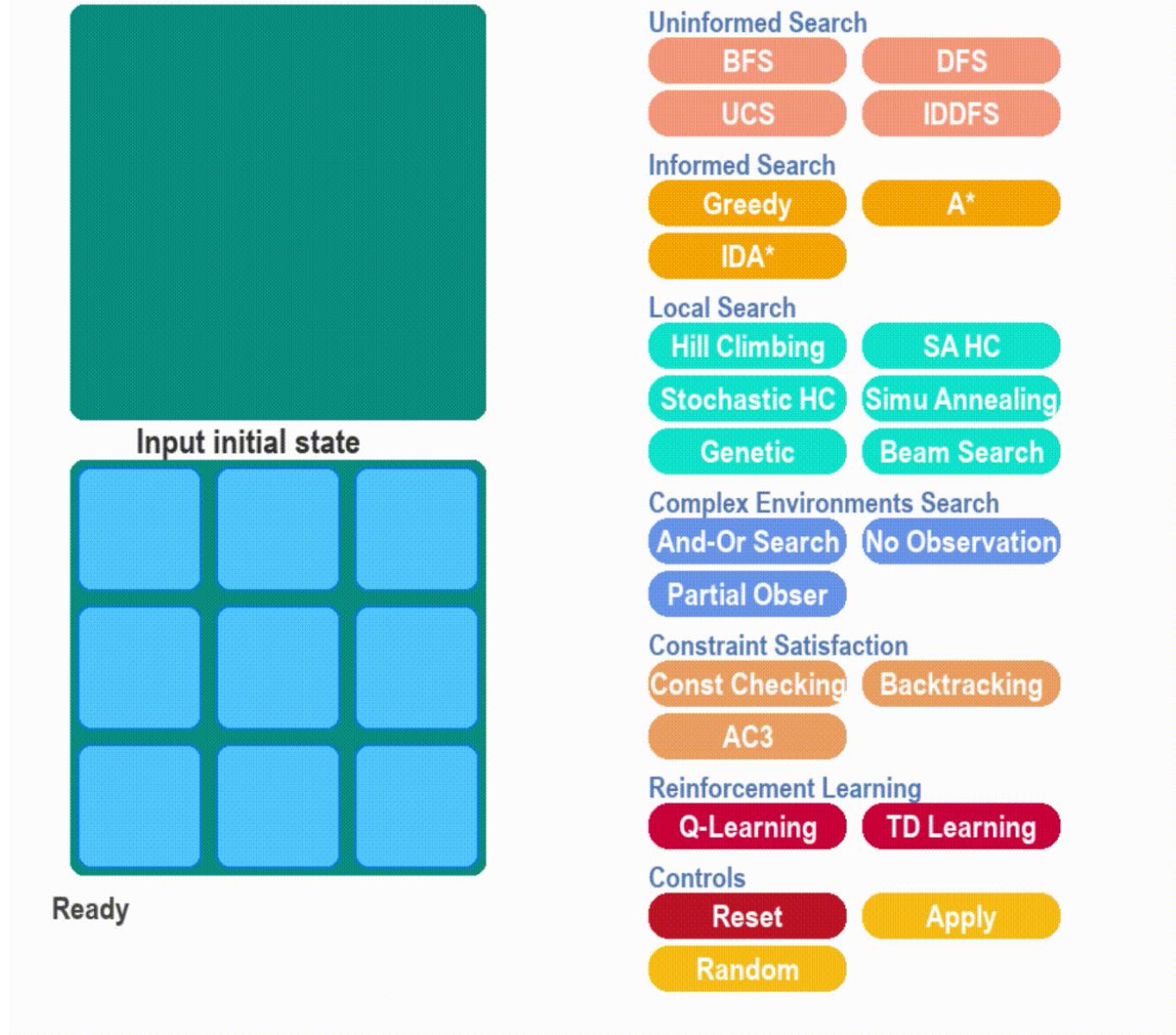
- **Mô tả:** Tìm kiếm bằng cách thử và quay lại khi rơi vào ngõ cụt

- **Minh họa:**



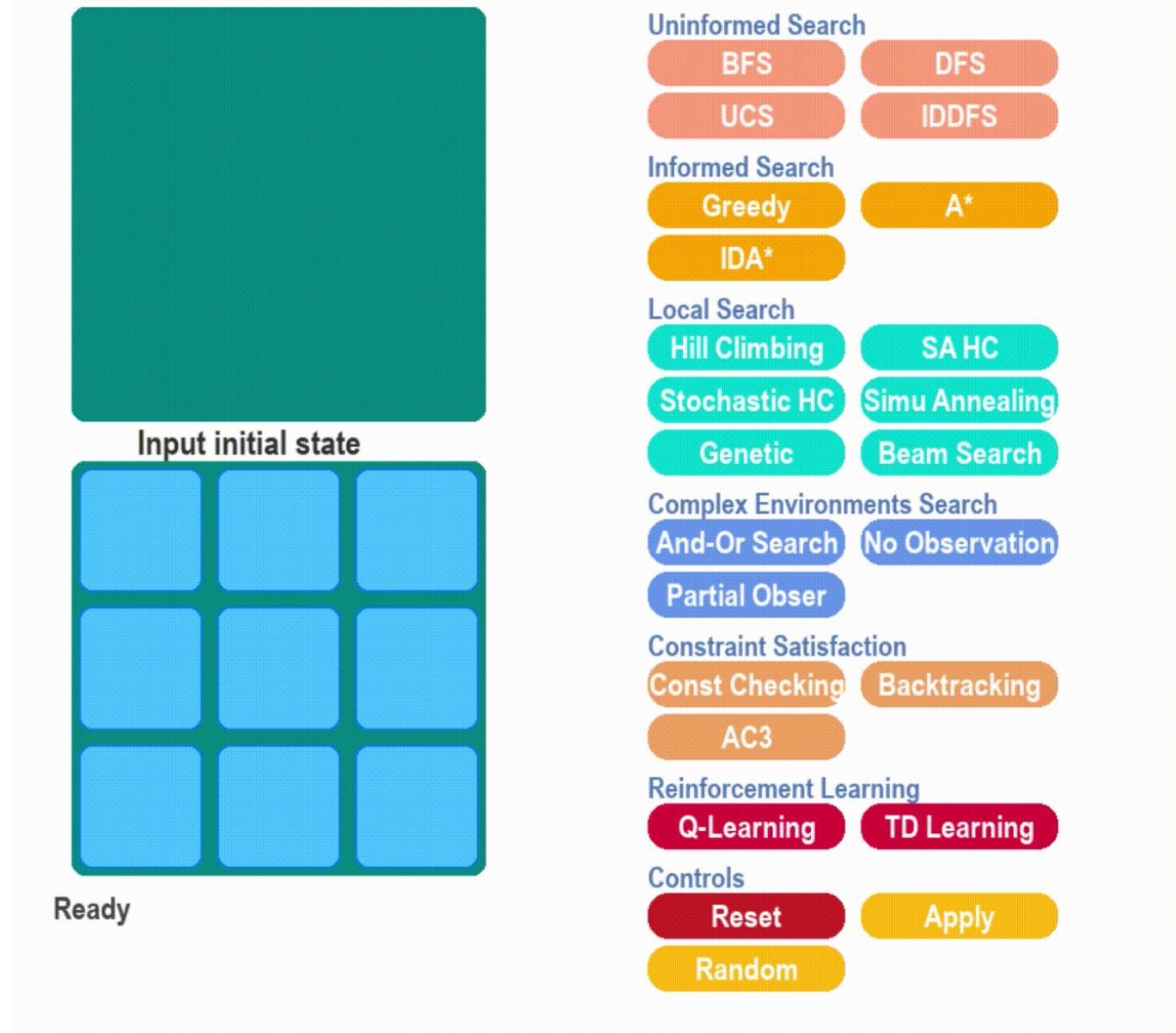
AC3 (Arc Consistency Algorithm #3)

- **Mô tả:** Thuật toán kiểm tra tính nhất quán giữa các ràng buộc
- **Minh họa:**



Constraint Checking

- **Mô tả:** Kiểm tra các ràng buộc trong quá trình tìm kiếm
- **Minh họa:**



So sánh hiệu suất:

Thuật toán	Thời gian giải (ms)	Bộ nhớ sử dụng	Đặc điểm
Backtracking	50-150	Thấp	Tìm tất cả các giải pháp khả thi
AC3	100-200	Trung bình	Cải thiện hiệu suất tìm kiếm bằng cách loại bỏ giá trị không hợp lệ
Constraint Checking	80-150	Thấp	Kiểm tra tính hợp lệ của các trạng thái

Nhận xét:

- **Backtracking** đơn giản và hiệu quả cho các bài toán nhỏ, nhưng có thể chậm với không gian trạng thái lớn
- **AC3** và **Constraint Checking** cải thiện hiệu suất tìm kiếm bằng cách loại bỏ sớm các giá trị không hợp lệ

2.6. Nhóm 6: Thuật Toán Học Tăng Cường (Reinforcement Learning)

Thành phần chính của bài toán tìm kiếm:

- **Trạng thái (State):** Là cấu hình hiện tại của môi trường, trong bài toán này chính là bảng 8-puzzle tại một thời điểm cụ thể.
- **Hành động (Action):** Tập các nước đi mà agent có thể thực hiện tại mỗi trạng thái, tương ứng với việc di chuyển ô trống lên, xuống, trái hoặc phải.
- **Phần thưởng (Reward):** Giá trị phản hồi từ môi trường sau mỗi hành động, thường được thiết kế để khuyến khích việc tiến gần hơn đến trạng thái đích và phạt nếu đi sai hướng.
- **Chính sách (Policy):** Chiến lược hành động tối ưu của agent, xác định hành động nào cần chọn trong mỗi trạng thái nhằm tối đa hóa phần thưởng tích lũy.
- **Solution (Lời giải):** Là chuỗi hành động được agent học thông qua quá trình tương tác với môi trường, giúp đưa từ trạng thái ban đầu đến đích theo chính sách tối ưu mà không cần duyệt toàn bộ không gian trạng thái.

Q-Learning 🎯

- **Mô tả:** Thuật toán học tăng cường để tìm chiến lược tối ưu
- **Minh họa:**

Temporal Difference (TD) Learning

- **Mô tả:** Thuật toán học giá trị trạng thái dựa trên sự khác biệt tạm thời (temporal difference) giữa giá trị hiện tại và giá trị kế tiếp.

- Minh họa:



So sánh hiệu suất:

Thuật toán	Thời gian giải (ms)	Bộ nhớ sử dụng	Đặc điểm
Q-Learning	1000ms - vài giây	Cao	Học dần dần chiến lược tối ưu
TD-Learning	500ms – 1500ms	Cao	Cập nhật nhanh, không cần mô hình môi trường

Nhận xét:

- **Q-Learning** có khả năng học và cải thiện hiệu suất theo thời gian, nhưng đòi hỏi nhiều tài nguyên
- **TD Learning** có thể học nhanh và nhẹ hơn do không cần lưu bảng Q đầy đủ, thích hợp với môi trường không xác định rõ mô hình.

3. Kết Luận

Kết quả đạt được

- Triển khai thành công 21 thuật toán tìm kiếm, bao gồm đầy đủ 6 nhóm: từ tìm kiếm không có thông tin, có thông tin, cục bộ, cho đến học tăng cường và môi trường ràng buộc phức tạp.
- Xây dựng một giao diện trực quan và dễ sử dụng bằng Pygame, giúp người dùng theo dõi trực tiếp quá trình giải quyết bài toán theo từng bước.
- Tổ chức lại toàn bộ nội dung thuật toán theo nhóm, kết hợp minh họa hình ảnh và bảng hiệu suất để trực quan và dễ tiếp cận.
- Phân tích chi tiết điểm mạnh và hạn chế của từng thuật toán khi áp dụng cụ thể vào trò chơi 8-puzzle.

Nhận xét tổng quát

- Các thuật toán tìm kiếm **không có thông tin** (BFS, DFS, UCS, IDDFS) phù hợp với bài toán nhỏ, ổn định nhưng dễ bị giới hạn bởi bộ nhớ hoặc thời gian trong các bài toán lớn hơn.
- Nhóm thuật toán tìm kiếm **có thông tin** (A*, IDA*, Greedy) mang lại hiệu quả vượt trội nhờ tận dụng heuristic, đặc biệt A* cho kết quả tối ưu một cách đáng tin cậy.
- **Local Search** cho thấy ưu thế về tốc độ và tiết kiệm bộ nhớ, tuy nhiên dễ bị rơi vào trạng thái cục bộ, trừ khi áp dụng các kỹ thuật như Simulated Annealing hoặc Genetic Algorithm.
- Nhóm thuật toán tìm kiếm trong **môi trường phức tạp** như And-Or Search, Partial Observable, No Observation thể hiện khả năng thích nghi cao trong điều kiện thiếu thông tin hoặc không chắc chắn.
- Các thuật toán tìm kiếm có **ràng buộc (CSP)** như Backtracking, AC3, Constraint Checking giúp nhanh chóng loại bỏ trạng thái không hợp lệ và làm nền tảng cho sinh lời giải ban đầu.
- Thuật toán học tăng cường **Q-Learning** mang lại một góc nhìn khác khi bài toán được học thông qua tương tác thay vì duyệt toàn bộ không gian trạng thái.

Hướng phát triển

- Cải thiện tốc độ và bộ nhớ thông qua tối ưu thuật toán và cấu trúc dữ liệu.
- Bổ sung các loại heuristic khác và cho phép người dùng tùy chọn trong giao diện.
- Mở rộng trò chơi sang phiên bản lớn hơn như 15-puzzle hoặc 24-puzzle.
- Tích hợp hệ thống giải thích trực quan từng bước tìm kiếm cho mục đích học tập và trình bày.

Cài Đặt và Chạy Game

Yêu Cầu

- Python 3.x 
- Thư viện **Pygame** (Cài đặt qua ):

```
pip install pygame
```

Cách Tải và Cài Đặt

1. Clone dự án về máy của bạn:

```
https://github.com/ThanhSangLouis/Eight\_Puzzle\_Solver.git
```

2. Chạy ứng dụng:

```
python -m eight_puzzle_solver.main
```

Hướng Dẫn Chơi 🎮

1. Chính Sửa Trạng Thái Ban Đầu:

- Nhấp vào các ô hoặc cuộn con lăn chuột để thay đổi giá trị. Ô trống sẽ là số 0.
- Bạn có thể nhấp chuột phải để thay đổi giá trị của ô trống từ 8 đến 0.

2. Chọn Thuật Toán:

- Chọn thuật toán từ danh sách để giải bài toán (ví dụ: BFS, A*, hoặc Simulated Annealing).
- Sau khi chọn thuật toán, ứng dụng sẽ bắt đầu giải quyết và hiển thị số bước đi và thanh tiến trình.

3. Reset ⏪:

- Bạn có thể nhấn "Reset" để quay lại trạng thái ban đầu của puzzle.

4. Hiển Thị Tiến Trình 📈:

- Số bước đi sẽ được cập nhật trong giao diện khi thuật toán đang chạy.
- Thanh tiến trình sẽ cho bạn thấy tiến độ giải bài toán.

❖ Cấu trúc dự án

```
Eight_Puzzle_Solver/
├── eight_puzzle_solver/
│   ├── __init__.py
│   ├── main.py          # Điểm vào chính của ứng dụng
│   ├── gui.py           # Xử lý giao diện đồ họa
│   ├── algorithms.py    # Các thuật toán giải 8-puzzle
│   └── utils.py         # Các hàm tiện ích
└── images/              # Hình ảnh
└── README.md            # Tài liệu dự án
```

🤝 Đóng góp

Mọi đóng góp đều được hoan nghênh! Nếu bạn muốn đóng góp, vui lòng:

1. Fork repository
2. Tạo branch mới (`git checkout -b feature/amazing-feature`)
3. Commit thay đổi (`git commit -m 'Add some amazing feature'`)
4. Push lên branch (`git push origin feature/amazing-feature`)
5. Mở Pull Request

👤 Tác giả

- **Họ tên:** Võ Thanh Sang
- **MSSV:** 23110301
- **Môn học:** Trí Tuệ Nhân Tạo
- **Trường:** Đại học Sư phạm Kỹ thuật TP.HCM (HCMUTE)

📞 Liên hệ

Thanh Sang - [@ThanhSangLouis](#)

Project Link: https://github.com/ThanhSangLouis/Eight_Puzzle_Solver

Built with ❤ by Thanh Sang

 Stars 0  Forks 0

[⬆ Về đầu trang](#)